Compiler State of the Union

Session 402

These are confidential sessions—please refrain from streaming, blogging, or taking pictures.

Compiler State of the Union

Chris Lattner
Manager and Architect
Compilers and Low-Level Tools

Compiler Intro and Landscape

iPhone OS 2.x:

GCC 4.0

iPhone OS 3.x:

GCC 4.2

Mac OS/X Leopard:

GCC 4.0

GCC 4.2

LLVM-GCC 4.2

Mac OS/X Snow Leopard:

GCC 4.0

GCC 4.2

LLVM-GCC 4.2

Clang

Blocks

Pass code and data as an expression

GCC 4.2
and Clang

- Simple example

```
int N = ...
dispatch_apply(count, ...
                    ^(size_t idx) {
  Array[idx] = Array[idx] + N;
});
```

Programming with Blocks and Grand Central Dispatch	Marina Wednesday 2:00PM

GCC 4.2

- Blocks
- 64-bit default
- Bug fixes

What Is LLVM?

- A new compiler strategy for Apple
- Complementing and eventually replacing GCC
- Open Source!

http://llvm.org/

Why Not Extend GCC?

- GCC
  - … does not get the most out of the hardware
  - … does not provide a great developer experience
  - … can not be reused to build new applications

It is time for something new!

How Is LLVM Different?

- Radically rethink how compilers are built and used
  - Build compilers as a set of reusable libraries
  - Support new applications with shared components
- Focused on performance and functionality
  - Fast compiles, great code generation
  - Use modern techniques and algorithms

Not Just Traditional Compilers

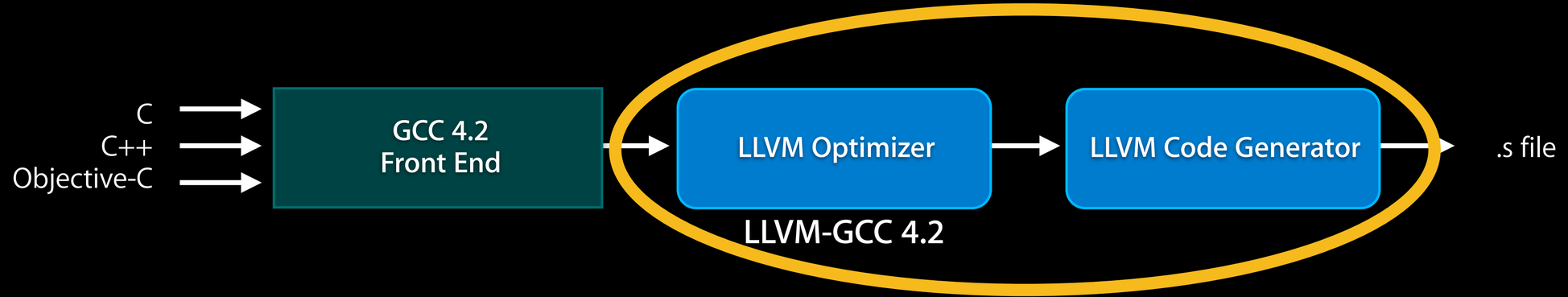
- LLVM and Clang are built to serve other clients
  - OpenGL, OpenCL, Xcode Static analyzer
  - LLVM-GCC and Clang Compiler

LLVM at Apple

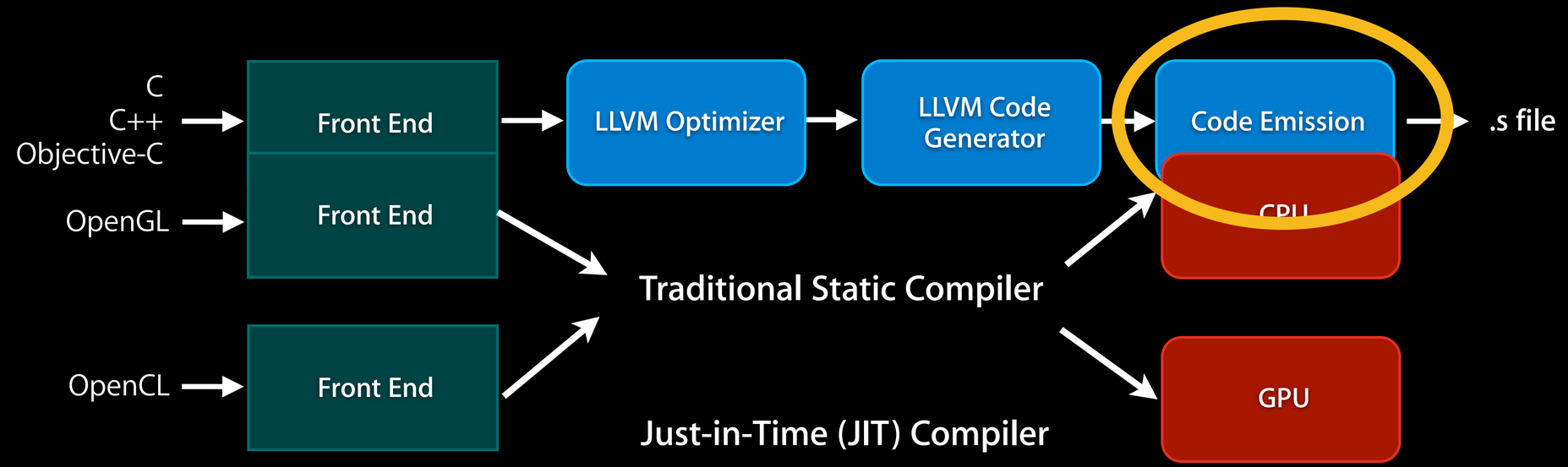
Evan Cheng
Manager and Technical Lead
Code Generator Team

GCC 4.2? LLVM-GCC 4.2? Clang?

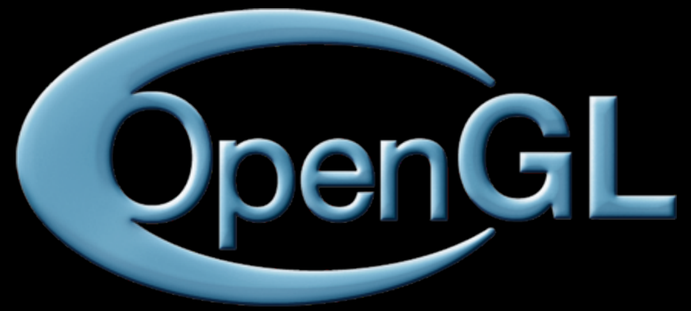
- Compiler has three pieces—parser, optimizer, code generation



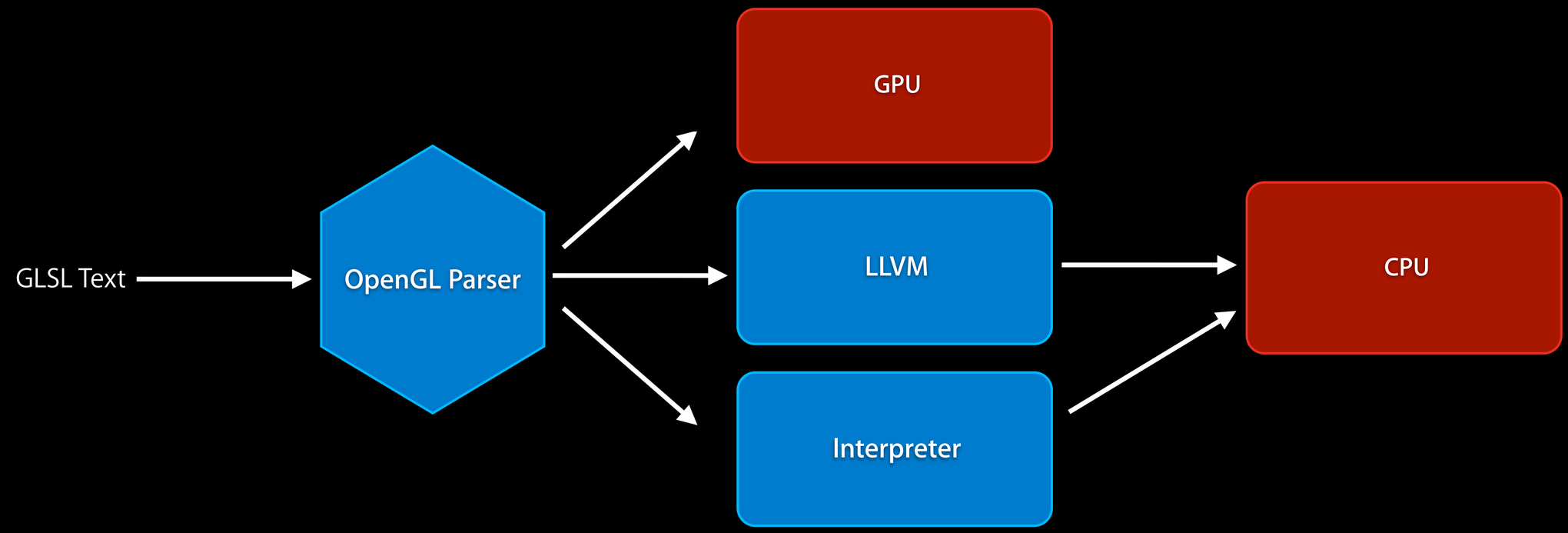
LLVM—High-Level Overview



OpenGL

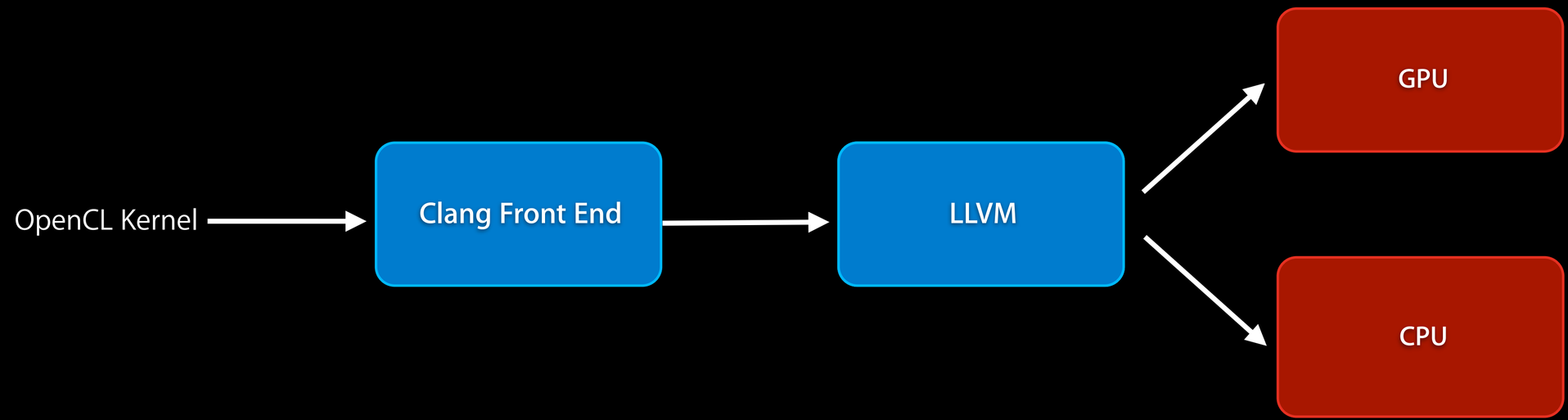


- Used custom JIT and interpreter pre-Leopard



OpenCL

- New in Snow Leopard
- Clang front end with LLVM optimizer and code generator



Speed up CoreImage by 25%!

Related Talks

Introduction to OpenCL	Marina Thursday 9:00AM

LLVM-GCC 4.2

LLVM-GCC Goals

- 100% compatible with GCC
- Feature parity
- Faster compilation
- Generate faster code

New LLVM-GCC Features This Year

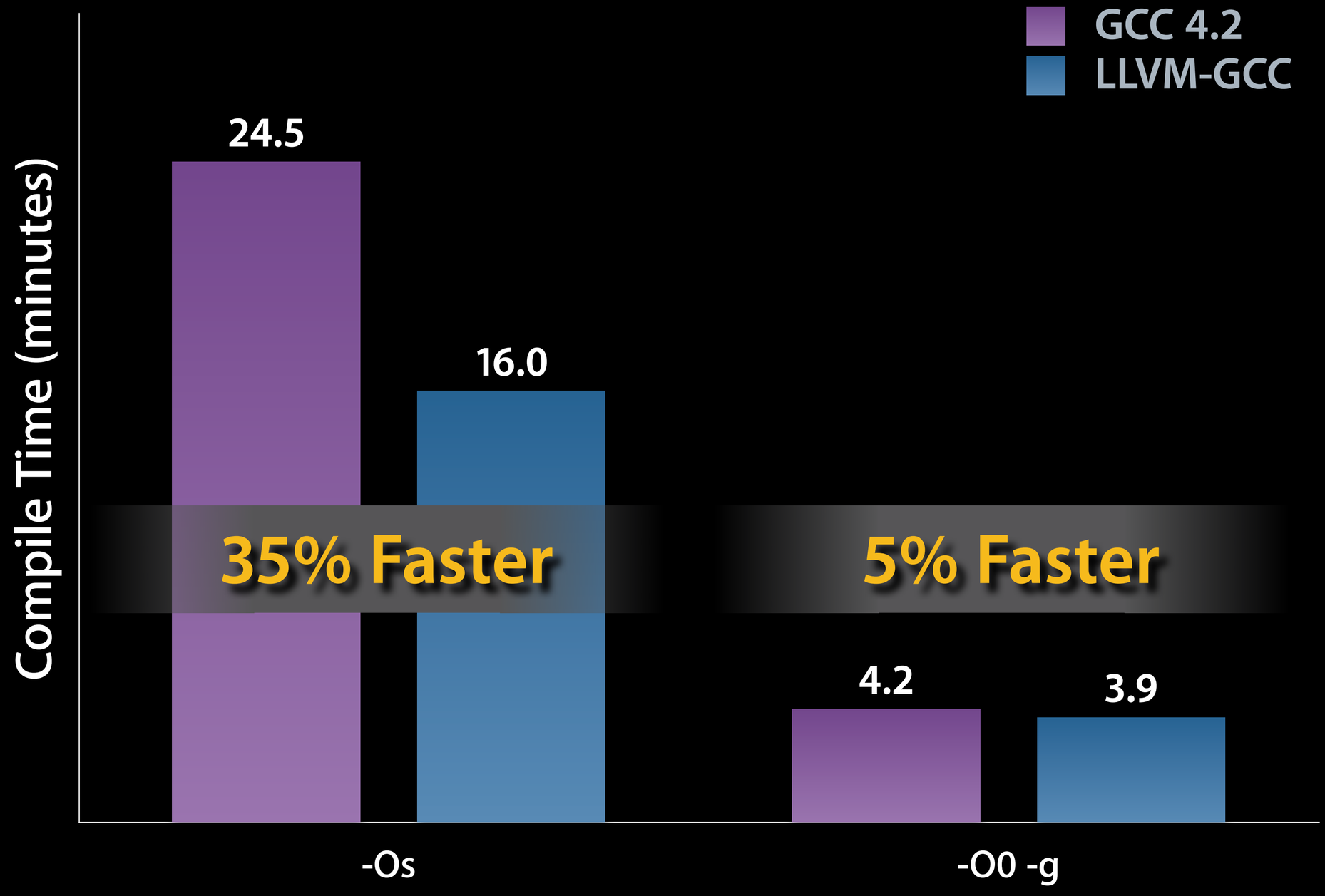
- Stack canaries
- Blocks!
- OpenMP
- Improved debugging

LLVM-GCC now supports all major GCC features!

Compile Time Performance

- Compare LLVM-GCC against GCC
- Building AppKit: A large Objective-C framework

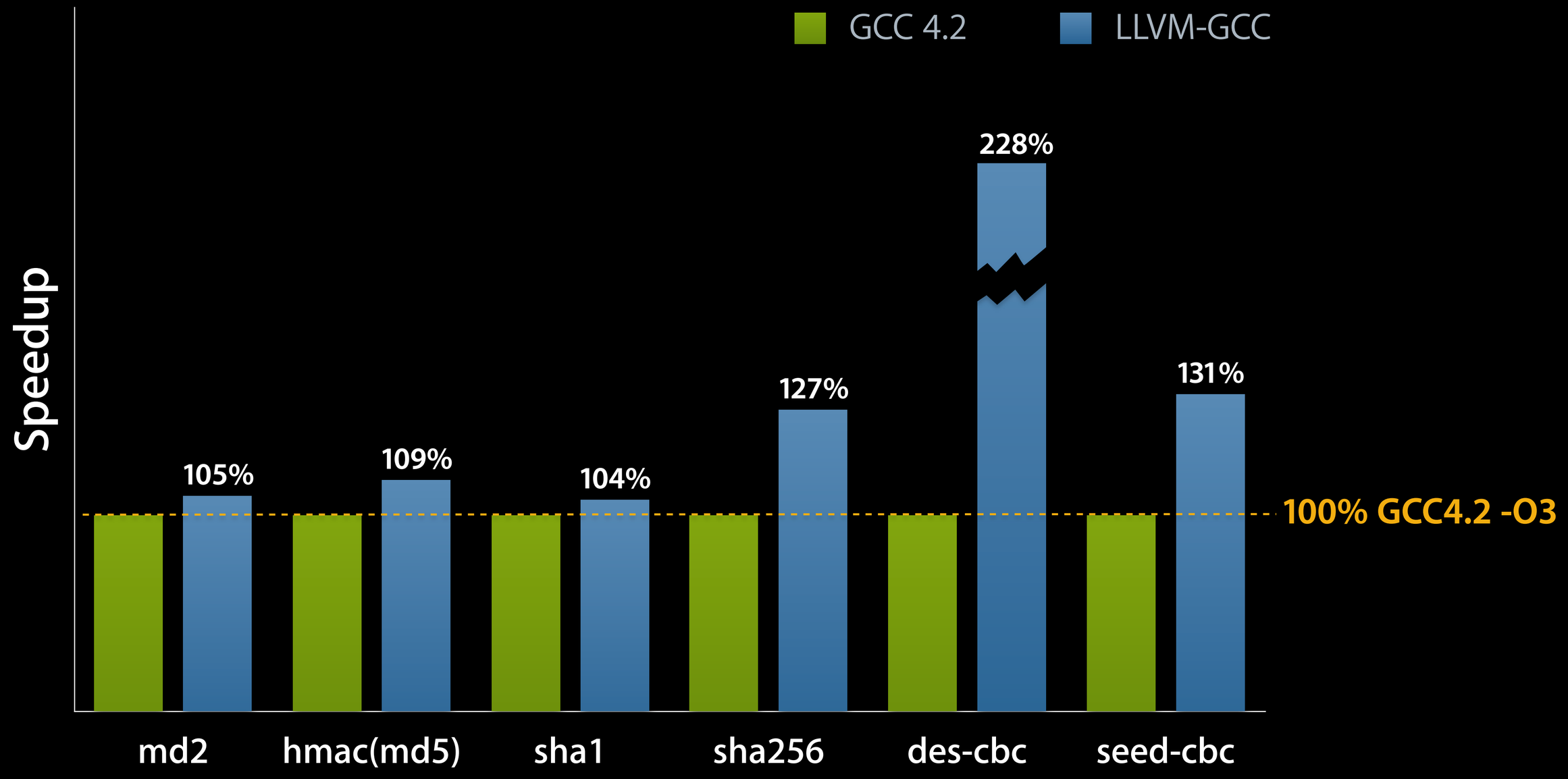
AppKit Framework Compile Times



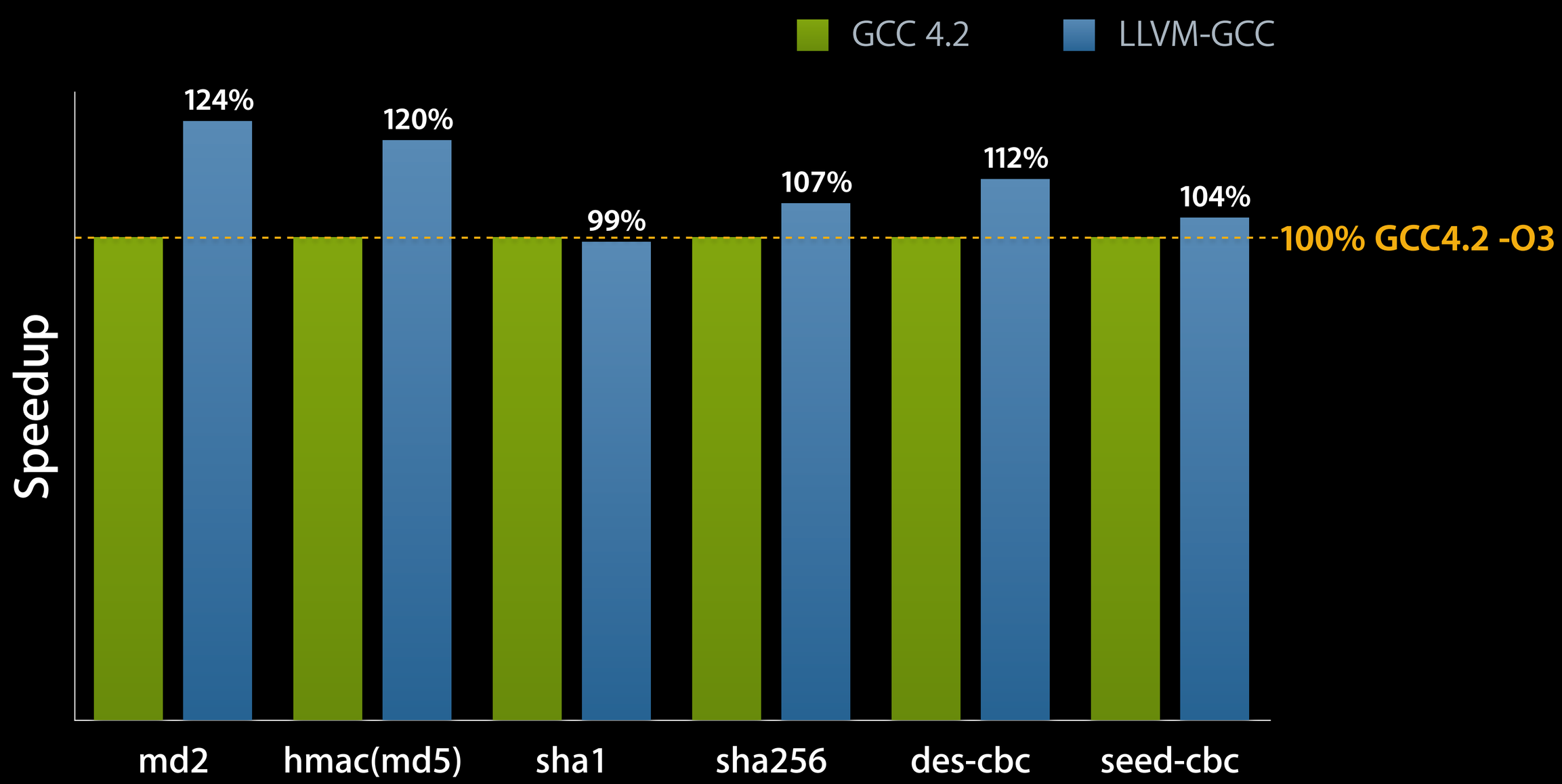
LLVM Codegen Performance

- Does LLVM-GCC generate faster code than GCC?
- Compare across a variety of benchmarks
- Focused on Intel CPUs
  - … both 32-bit and 64-bit

OpenSSL 32-Bit



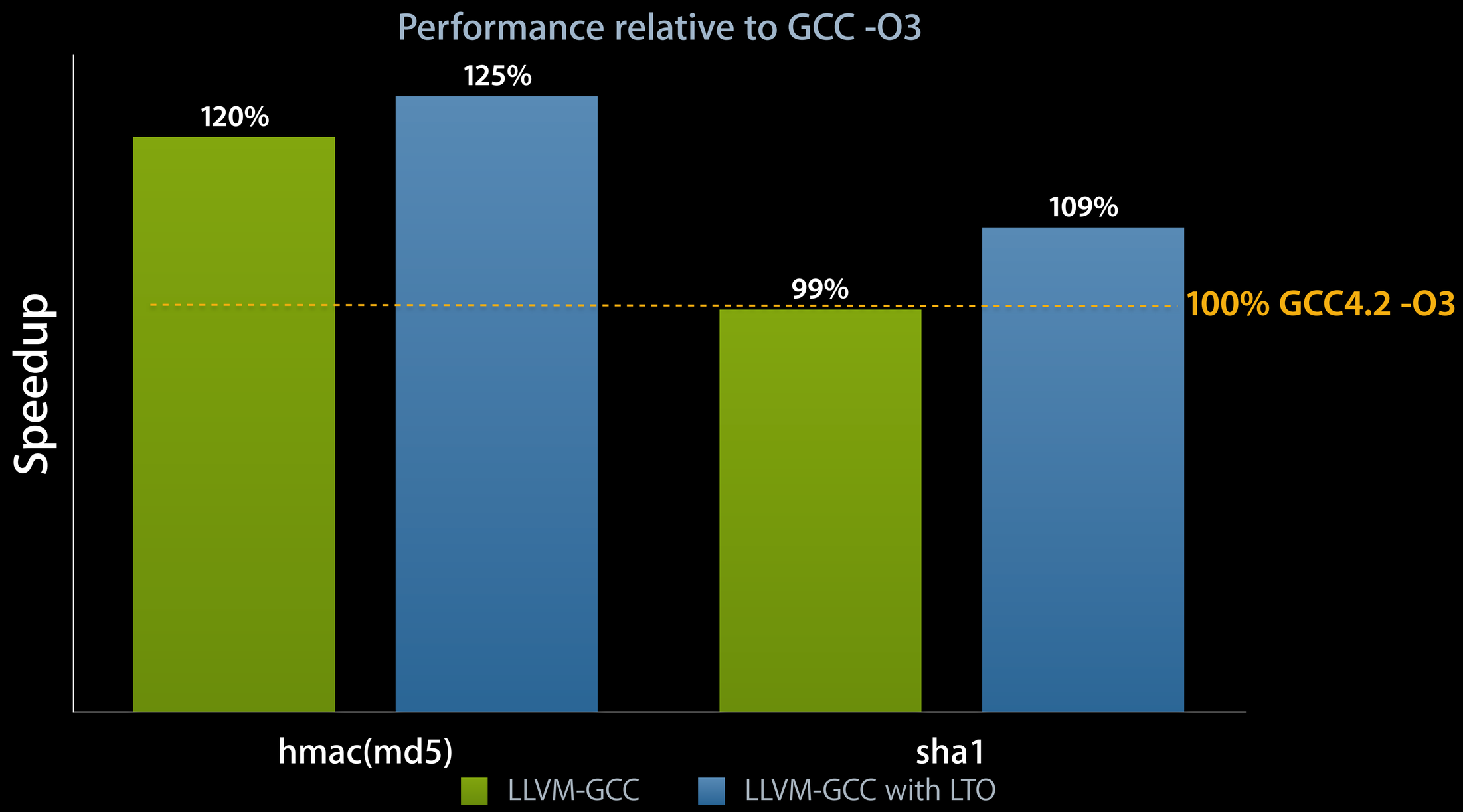
OpenSSL 64-Bit



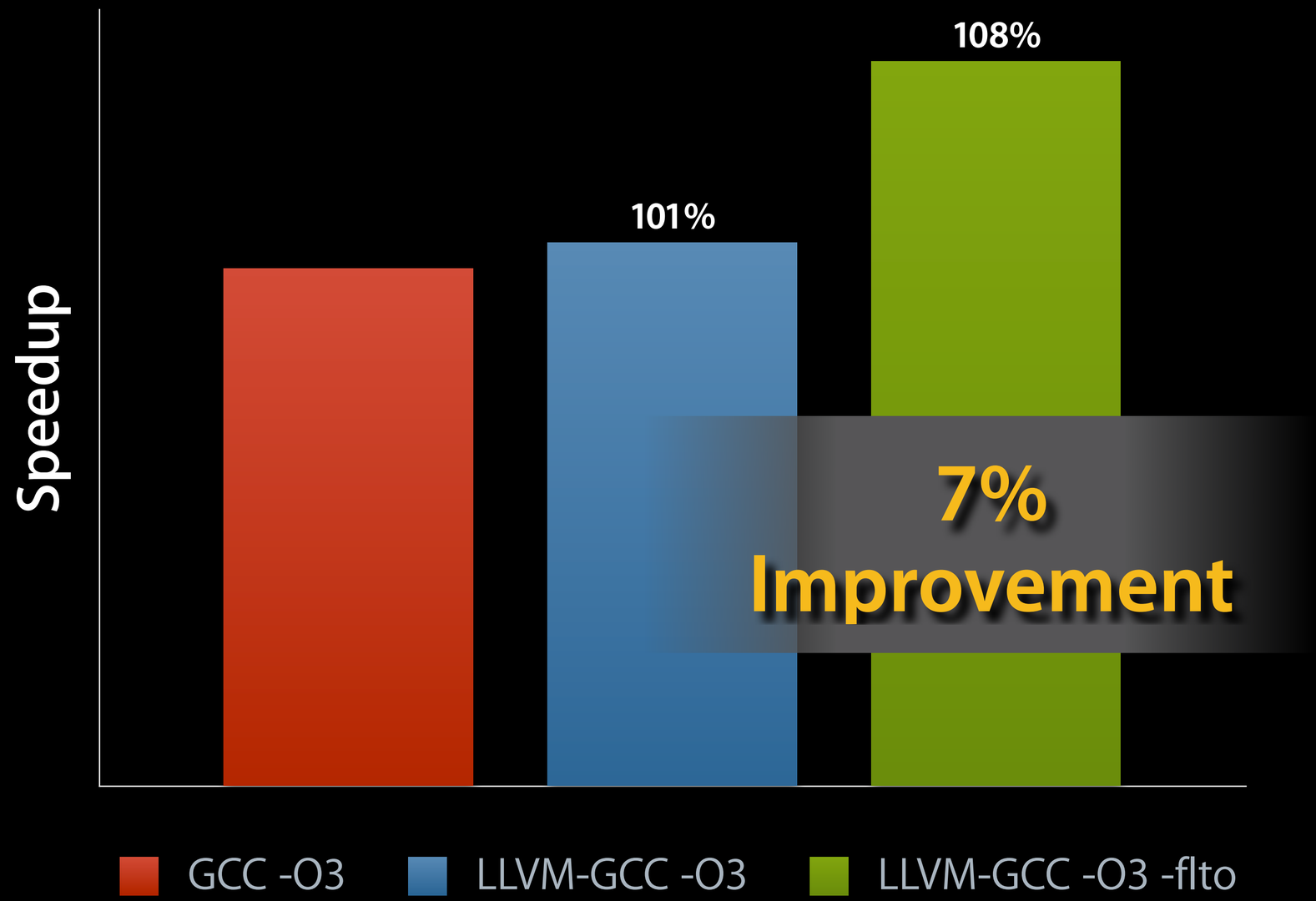
Link Time Optimization

- Aggressive optimization across files

OpenSSL 64-Bit with LTO



JPEG Decoding Time



Enabling Link Time Optimization

- Xcode

- Command Line

```
/Developer/usr/bin/llvm-gcc -O3 -flto   foo.c -o foo.exe
/Developer/usr/bin/llvm-gcc -O4         foo.c -o foo.exe
```

LLVM-GCC Built Snow Leopard Projects

- CoreData
- OpenSSL
- Java HotSpot

LLVM-GCC vs GCC 4.2

- Features
  - OpenMP, stack canaries, … all supported
  - LLVM only: Link Time Optimization
- Generated code
  - 32-bit: Significantly better
  - 64-bit: On par or better
- Compilation speed
  - “Release” mode: Usually over 30% faster
  - “Debug” mode: Slightly faster, 3%–7%

Clang!

Chris Lattner

What Is Clang?

Clang Libraries

- C and Objective-C front end
- LLVM design approach

Clang Compiler

- Clang front end and LLVM back end

Clang Open Source Project

http://clang.llvm.org/

Why Do We Care?

- We want an amazing compiler experience!
  - Fast compile times
  - Fast generated code
  - Great user features

Clang and LLVM are the Apple compiler strategy!

Clang Demo

Clang Topics

- ✓ Intro and Demo
- Compatibility
- Compile times
- Clang goodies
- Wrap up

Clang C++ Support

Clang C++ support is a critical for Apple

- … and we are committed to implementing it

Clang 1.0 has no C++ support

- Transparently falls back to llvm-g++ if you try to use it

Work is well underway, join us!

- http://clang.llvm.org/

Command Line Compatibility

Command line compatible with GCC 4.2

- Run it directly
  - **/Developer/usr/bin/clang hello.c -o hello**
- Build a makefile project
  - **make CC=/Developer/usr/bin/clang**
    CXX=/Developer/usr/bin/llvm-g++

Source Compatibility with GCC

Differences slow compiler transitions

Objective-C Compatibility

- Extreme compatibility with Objective-C code
- Intentionally broke some specific Objective-C idioms
  - Incorrect and already deprecated constructs
- If you depend on these, fix your code
  - We don’t plan to add them
  - Static analyzer won’t work

Unsupported Idioms

Cast of Super

```
[(myinterface*)super add: 4]
```

→

```
[super add: 4]
```

Cast of L-Value

```
(int*)addr = val;
```

→

```
addr = (float*)val;
```

sizeof Objective-C interfaces (64-bit)

```
sizeof(NSArray)
```

→

```
class_getInstanceSize([NSArray class])
```

Gotcha—Clang Defaults to C99!

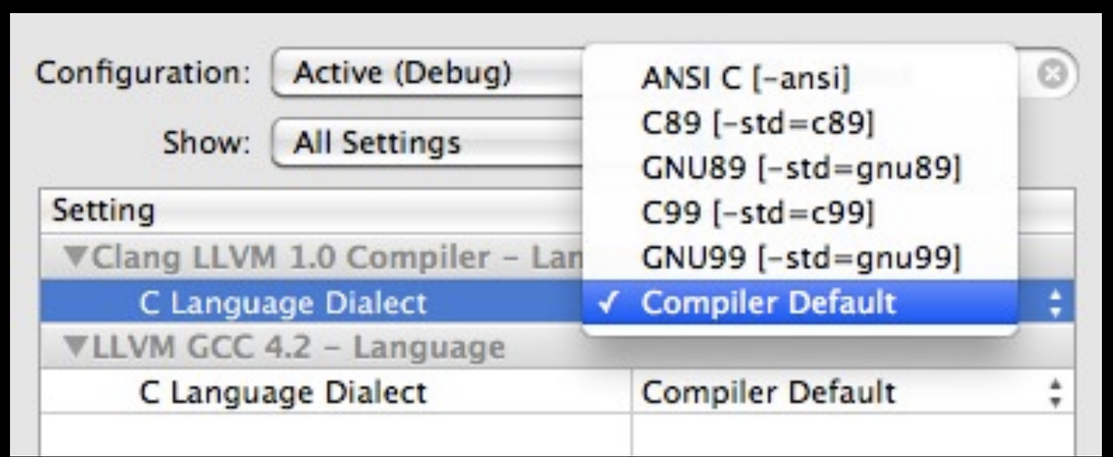
- C99 is mostly a transparent extension to C89
  - … but watch out for link failures due to “inline” difference

```
inline void foo() { ... }
```

→

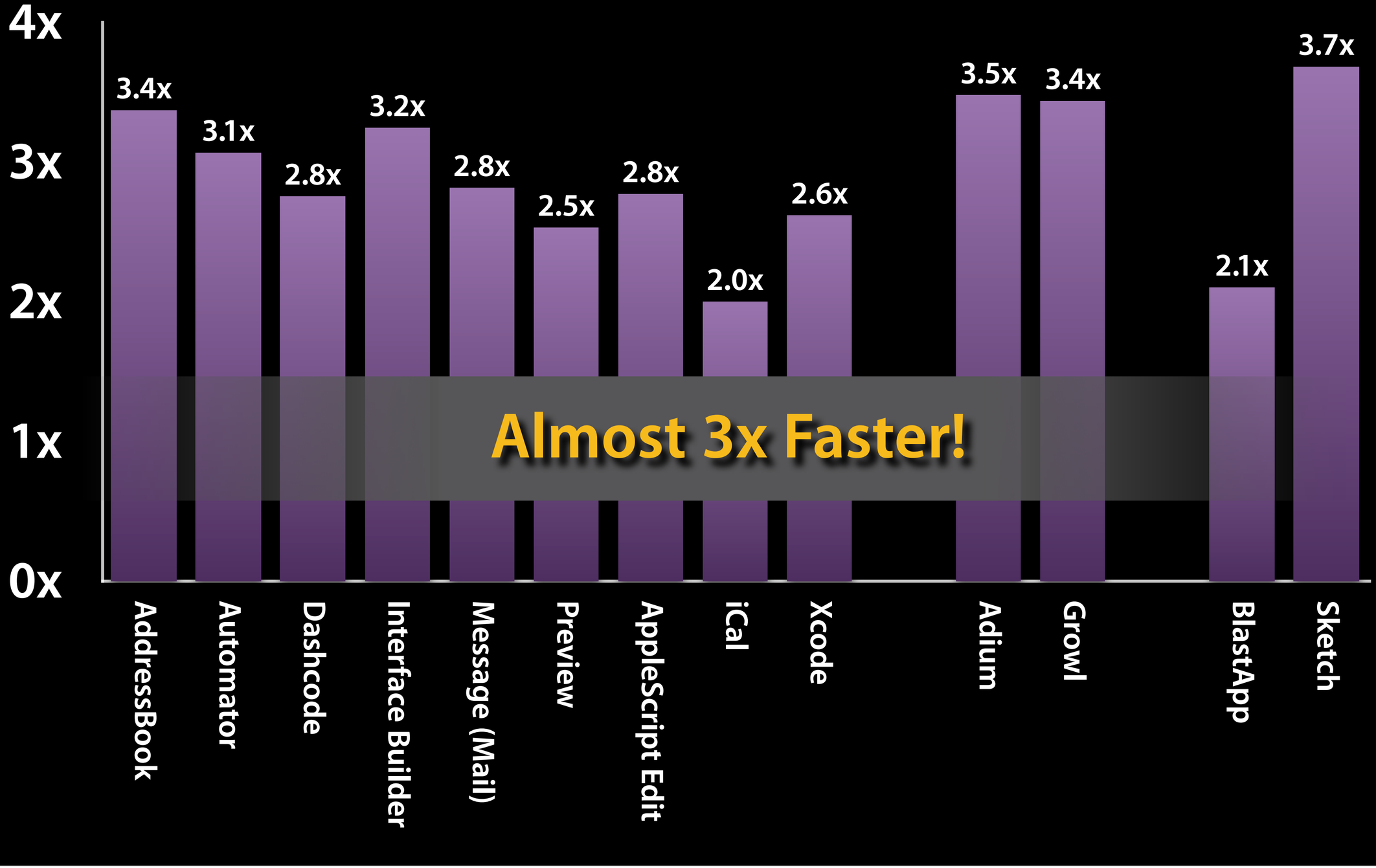
```
static inline void foo() { ... }
```

- If you really want C89
  - Set “C Language Dialect” build setting in Xcode

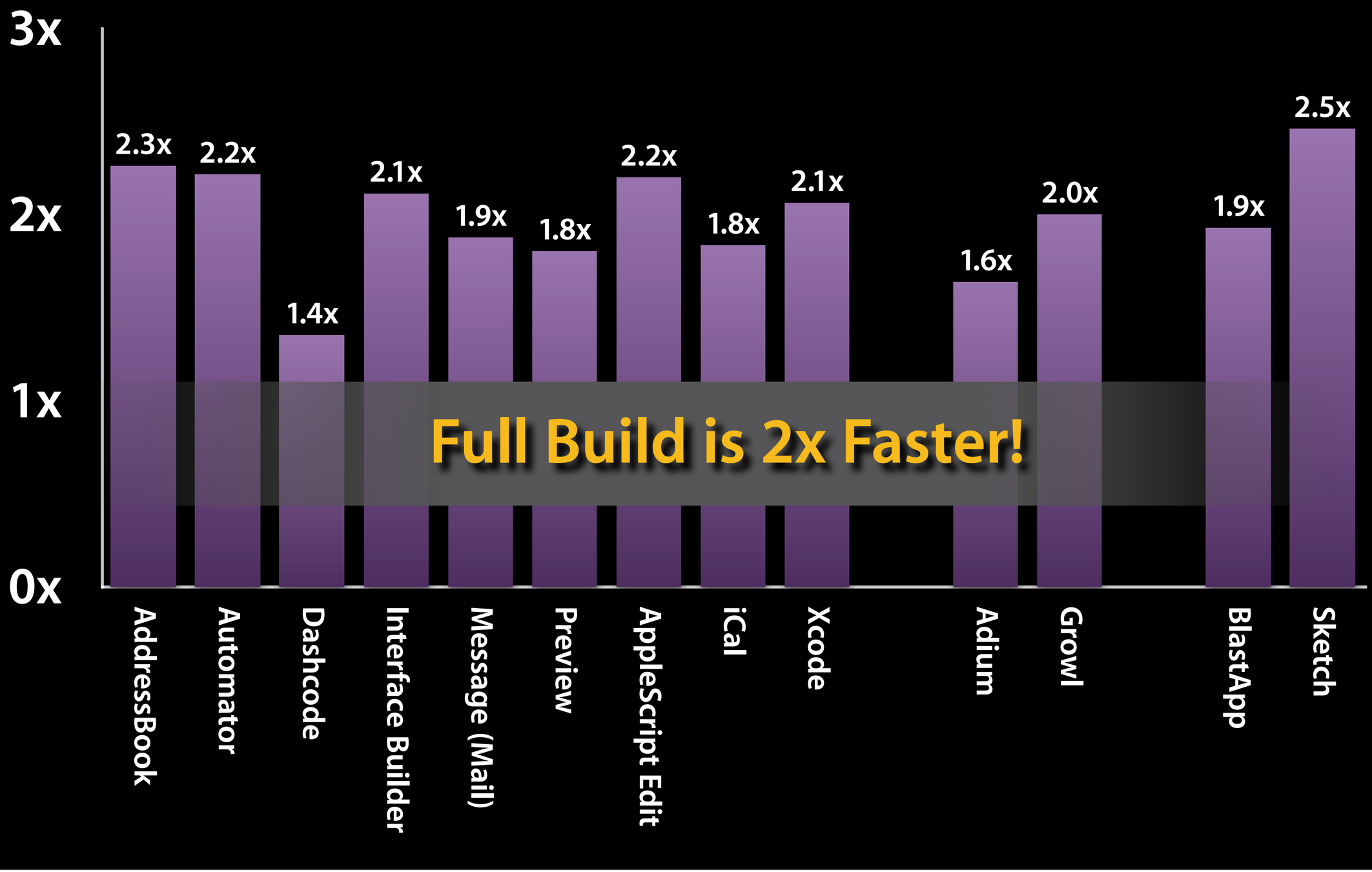


Compile Time Speed

Clang Compiler Speedup (Debug Config)



Full Build Speedup (Debug Config)



Why Is Clang Faster Than GCC?

Thousands of microscopic engineering details

- Better algorithms
- Less memory

Separation of Front End and Back End

GCC coevolved its C front end with its back end

- Tangled interconnections and interdependencies

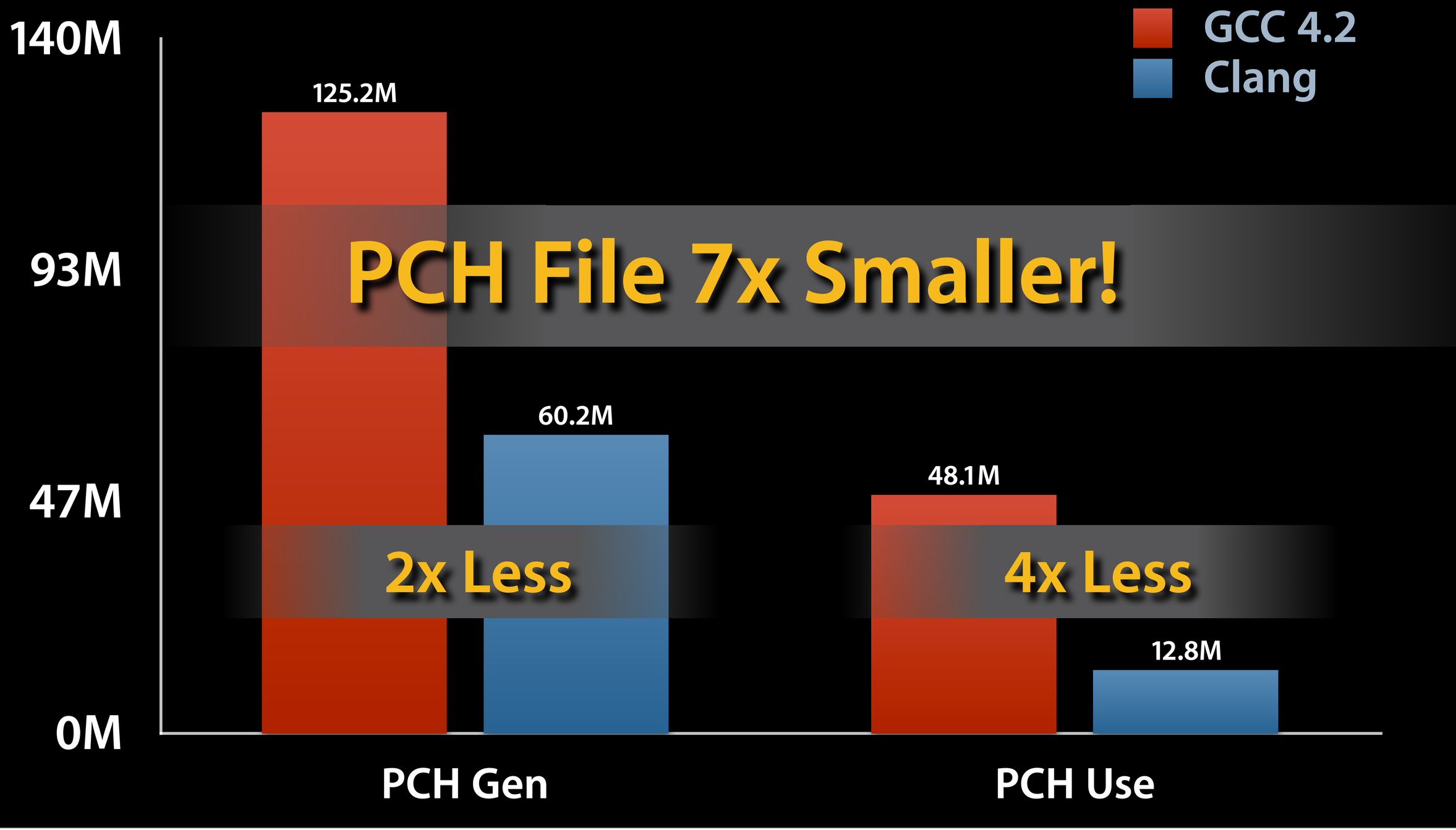
LLVM evolved independently of Clang

- Clang front end purposely kept independent of LLVM back end

Why does this matter?

- GCC wastes a lot of time and memory on codegen of headers
- … declarations that are never actually used in the .m file!

Precompiled Headers and Memory Use



Clang Goodies

Error and Warning Improvements

- "Diagnostics" are error and warning reports
  - Compiler detects something is wrong
  - Tries to guess why and explain it

- GCC diagnostics are often not helpful
  - Confusing, poorly worded, not precise

```
NSString *P = @"good stuff";
              Expected '=', ',', ';', 'asm' or '__attribute__' before '*' token
```

Clang should do better!

Better Diagnosis of the Problem

GCC 4.2

```
NSString *P = @"good stuff";
              Expected '=', ',', ';', 'asm' or '__attribute__' before '*' token
```

Clang

```
NSString *P = @"good stuff";   Unknown type name 'NSString'
```

```
test.m:4:1: error: unknown type name 'NSString'
  NSString *P = @"good stuff";
  ^
```

Range and Location Information

GCC 4.2

```
X = X + func(X ? ((SomeA.F + 40) + SomeA) / 42 +
                 Invalid operands to binary +
```

Clang

```
X = X + func(X ? ((SomeA.F + 40) + SomeA) / 42 + S
                 Invalid operands to binary expression
```

```
t.m:29:36: error: invalid operands to binary expression ('int' and 'struct A')
   X = X + func(X ? ((SomeA.F + 40) + SomeA) / 42 + SomeA.F : Ptr->F);
                     ~~~~~~~~~~~~~~ ^ ~~~~~
```

Macro Expansion Information

GCC 4.2

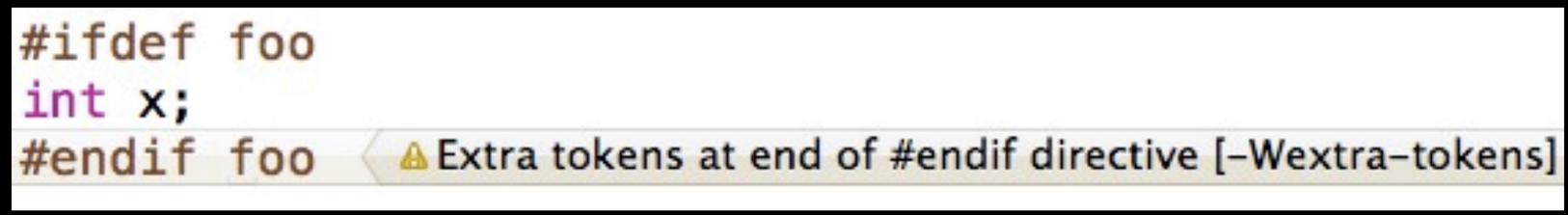
```
#define MAX(A, B) ((A) > (B) ? (A) : (B))

  X = MAX(X, *Ptr);                    Invalid operands to binary >
```

Clang

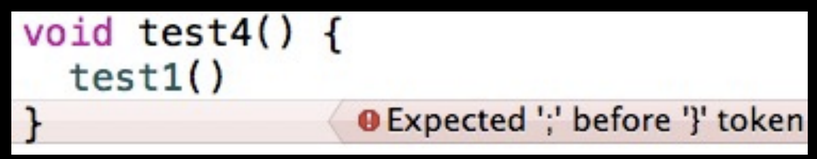
Invalid operands to binary expression ('int' and 'struct A')
Instantiated from: /Users/sabre/Desktop/testapp/main.m

```
t.m:48:7: error: invalid operands to binary expression ('int' and 'struct A')
   X = MAX(X, *Ptr);
       ^~~~~~~~~~~~
t.m:43:24: note: instantiated from:
 #define MAX(A, B) ((A) > (B) ? (A) : (B))
                    ~~~ ^ ~~~
```

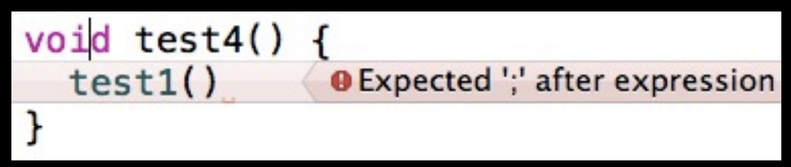
Other Great Refinements




```
t.c:4:8: warning: extra tokens at end of #endif directive [-Wextra-tokens]
#endif foo
       ^
```




```
t.c:53: error: expected ‘;’ before ‘}’ token
```




```
t.c:52:10: error: expected ';' after expression
test1()
       ^
```

Command Line Warning Control

Command Line Option	Effect
-Wfoo	Enable warning foo
-Wno-foo	Disable warning foo
-Werror=foo	Enable warning foo, emit as an error
-Werror	Emit all warnings as errors
-Werror -Wno-error=foo	Emit all warnings as errors, but keep foo a warning

Pragma Control for Warnings

```
#pragma clang diagnostic ignored “-Wfoo”
```

```
#pragma clang diagnostic warning “-Wfoo”
```

```
#pragma clang diagnostic error “-Wfoo”
```

```
#pragma clang diagnostic fatal “-Wfoo”
```

Vector Extensions

Declaring Extended Vectors

```
typedef float __attribute__((__ext_vector_type__(4))) float4;
v1 = (float4){-7,-7,-7,-7};                          // v1 = {-7, -7, -7, -7 }
v2 = (float4){0, 1, 2, 3};                           // v2 = { 0,  1,  2,  3 }
```

Basic Vector Operations

```
v3 = v1 + v2*v2;                                     // v3 = {-7, -6, -3, 2 }
v4 = *(float4*)Ptr;                                  // load,store, etc
```

Natural Element Access and Shuffles

```
f  = v1[3];                                          // f = -7
v5 = v2.odd;                                         // v5 = { 1, 3 }
v6 = v2.wzyx;                                        // v6 = { 3, 2, 1, 0 }
v7 = __builtin_shufflevector(v1, v2, 1, 6, 4, 5);    // v7 = {-7, 2, 0, 1 }
```

Introduction to OpenCL	Marina Thursday 9:00AM

Can I Actually Use Clang?

Clang really works!

- Built on the stable and well-tested LLVM code generator
- Err on the side of rejecting code—not accepting and breaking it

Broad testing on all kinds of crazy code

- Wide range of internal Apple code
- Countless open source UNIX and Mac apps, Linux kernel, etc.
- FreeBSD kernel and much of entire userspace

Clang-Built Apps in Snow Leopard

In addition to building code, we are even shipping it!

Two apps (Xcode and xcodebuild)

18 frameworks, 20 plug-ins

17 Xcode projects

2.5 million lines of code (~5% is Objective-C++)

“I've never seen a transition to a new compiler go as smoothly as this.

When I say never, it's somewhat authoritative. :-)"

Chris Espinosa
Xcode Lead
Apple Employee #8

Clang Static Analyzer

Analyzer Xcode Integration

```
NSObject *objectID = 0;

for (NSUInteger i=0; i < count; ++i) {          Looping back to the head of the loop

    NSObject *object = [trackedElements objectAtIndex:i];

    if ([object isMemberOfClass:[NSString class]])
    {
        objectID = [[NSString alloc] initWithString:aString];
                    Method returns an Objective-C object with a +1 retain count (owning reference)
    }

    if (objectID != nil)
    {
        [objectID release];                     Object released
    }                                           Reference-counted object is used after it is released
}
```

Improving Your Application with the Xcode Static Analyzer

North Beach
Tuesday 5:00PM

Which Compiler Should I Use on the Mac?

Fast Compiles
Fast Generated Code
Great User Experience

Clang!

Fast Optimized Compiles
Fast Generated Code
Full GCC Compatibility

LLVM-GCC 4.2

GCC 4.2

Stable Compiler from
Leopard + Blocks

GCC 4.0

Compatible with
Tiger SDK

More Information

Michael Jurewitz
Developer Tools Evangelist
jurewitz@apple.com

Related Sessions

Programming with Blocks and Grand Central Dispatch	Marina Wednesday 2:00PM
Introduction to OpenCL	Marina Thursday 9:00AM
Improving Your Application with the Xcode Static Analyzer	North Beach Tuesday 5:00PM

Related Labs

Compiler Lab	OS X Lab D Wednesday 9:00AM
Xcode Static Analyzer Lab	OS X Lab D Thursday 3:30PM

Q&A

WWDC Session Survey

You can rate your sessions on the WWDC Attendee Site.

Survey forms are located on each session description page at:

developer.apple.com/wwdc/attendee

WWDC09